

Universidad Tecnológica de Pereira

Reconocimiento Óptico de Caracteres en Placas Vehiculares haciendo uso de Redes Neuronales Convolucionales

Anderson Steven Mosquera Mosquera
Juan Pablo Martínez Rendón



Pereira, Risaralda

Reconocimiento Óptico de Caracteres en Placas Vehiculares haciendo uso de Redes Neuronales Convolucionales

Facultad de Ingenierías
de la Universidad Tecnológica de
Pereira.

Presentado por
Anderson Steven Mosquera Mosquera
Juan Pablo Martínez Rendón
en Pereira, Risaralda

Colombia, Noviembre 2018

Director: Ramiro Andrés Barrios Valencia
MSc. Ingeniería de Sistemas y Computación

Dedicatoria

“A mi familia, por su apoyo, sacrificios y la confianza depositada, ya que esto me ha permitido luchar por mis sueños y de esta forma culminar mis objetivos de la mejor manera”.

Anderson Steven Mosquera Mosquera

“Especial agradecimiento a mis padres por su amor, apoyo, trabajo y sacrificio durante todos estos años, a mi hermano y hermana por estar presente, acompañándome en esta etapa”.

Juan Pablo Martínez Rendón

Índice general

1. Introducción	3
2. Formulación del Problema	5
3. Justificación	7
4. Objetivos	9
4.1. Objetivo general	9
4.2. Objetivos específicos	9
5. Marco Referencial	11
5.1. Marco Teórico	11
5.2. Antecedentes	11
5.3. Bases Teóricas	12
5.4. Marco Geográfico	13
5.5. Marco Jurídico	14
5.6. Marco Conceptual	15
6. Diseño Metodológico	17
7. Delimitación del alcance	19
8. Implementación	21
8.1. Levantamiento de Información	21
8.1.1. Materiales y Métodos	21
8.1.2. Arquitectura Red Neuronal Convolutiva	22
8.1.3. Dataset	23
8.1.4. Backpropagation	24
8.2. Entrenamiento	24
8.3. Algoritmo	25
8.3.1. Imagen inicial	25
8.3.2. Pre-procesamiento	26
8.3.3. Inferencia	28
9. Pruebas	30
10. Resultados	32
11. Problemas Encontrados	36
11.0.1. Conjunto de datos en Pytorch	36
11.0.2. Orden de las regiones de interés	36
12. Conclusiones	38
Futuros trabajos	39
Bibliografía	40

Anexos	43
Participantes	49

1. Introducción

La visión por computador es una rama de la inteligencia artificial, que tiene como objetivo comprender los elementos o características de una escena o imagen del mundo real, todo esto basado en poder automatizar algunos procesos o simplemente permitir que un computador pueda realizar tareas que normalmente ejecuta un ser humano.

Por tal motivo surge la posibilidad de automatizar algunos procesos que para las personas son repetitivos o requieren una concentración extrema, principalmente en temas asociados a vigilancia. Por consiguiente nace la necesidad de construir herramientas que faciliten las labores operativas y poder suplir la presencia de un ser humano en algunos procesos.

Continuando el hilo de la información y haciendo uso de un enfoque a problemas de control de tráfico aparece una herramienta bastante versátil debido a sus diferentes formas de realizarse como lo son los sistemas ALPR (Automatic License Plate Recognition) que permiten satisfacer las razones anteriormente mencionadas y además poder atacar esta problemática con herramientas de visión artificial como lo son las redes neuronales convolucionales.

2. Formulación del Problema

En la actualidad las Redes Neuronales Convolucionales se han convertido en el método más usado para solucionar problemas que se consideraban conceptualmente complejos para un ordenador, ya que era necesario razonar para obtener su solución, éstas redes logran aportar soluciones a esta clase de problemas gracias a la capacidad de emular el funcionamiento del cerebro humano en áreas como el reconocimiento de objetos en imágenes, clasificación automática, aprendizaje de un lenguaje, etc.

Por tal motivo dentro de la Universidad Tecnológica de Pereira y más específicamente en el Grupo de Investigación Sirius, nace la necesidad de experimentar con el funcionamiento de dichas redes, con el fin de construir herramientas alternativas a las ya construidas dentro de la rama de investigación ITS (Sistemas Inteligentes de Transporte) que contribuyan a las investigaciones realizadas al control de tráfico de la ciudad.

3. Justificación

En el grupo de investigación Sirius de la Universidad Tecnológica de Pereira, perteneciente al programa de Ingeniería de Sistemas y Computación, existe una rama de investigación orientada a Sistemas Inteligentes de Transporte (ITS), en dicha rama se llevó a cabo la implementación de un algoritmo de conteo y clasificación de vehículos, que funciona a través de cámaras de seguridad colocadas en intersecciones de tráfico semaforizadas de la ciudad de Pereira, uno de sus objetivos es identificar el momento en que un auto incurre en una infracción de tránsito, específicamente cuando un vehículo transita por el carril exclusivo del sistema masivo de transporte Megabús o en el momento que un vehículo cruza un semáforo en rojo, inmediatamente captura por medio de una fotografía el momento en que ocurre la infracción. A partir de esta fotografía surge la necesidad de identificar los dígitos alfanuméricos correspondientes a la placa del automóvil asociado a la infracción. Por tal motivo desde el grupo de investigación se implementó una solución que consiste en el desarrollo de un algoritmo, utilizando métodos clásicos de visión por computador, como lo son SIFT (Scale-Invariant Features Transform) y SURF (Speeded-UP Robust Features) que permiten extraer características específicas dentro de una imagen, en este caso en particular la extracción de información de una placa vehicular.

La solución anteriormente mencionada se implementó sobre una mini-pc intel NUC, que cuenta con un procesador Intel Core i7 de 5ta generación, el cual solo permite ejecución de forma serial del algoritmo en CPU, ya que no cuenta con una unidad de procesamiento gráfico (GPU) que es indispensable si se desea utilizar computación paralela. El hecho de ejecutar el algoritmo solo en CPU impide la automatización de la detección de placas vehiculares, ya que será un proceso muy lento y evitará que sea realizado en tiempo real, como normalmente debería practicarse cualquier actividad relacionada con la visión por computador. Al querer realizar el proceso en tiempo real y de forma eficiente nace la necesidad de utilizar computación de alto rendimiento (High Performance Computing) y unidades de procesamiento gráfico, para aprovechar los recursos al máximo y poder construir una solución alterna a la mencionada anteriormente se hará uso de redes neuronales convolucionales.

La aplicación de las redes neuronales convolucionales es realizada en matrices bidimensionales las cuales se podrían asociar con imágenes, ya que una imagen está representada como una matriz de píxeles, tal motivo hace que éstas redes sean una buena elección para la solución; Usar redes neuronales convolucionales es un proceso costoso (computacionalmente hablando) entonces es necesario el uso de computación de alto rendimiento, con el fin de acelerar el algoritmo de forma paralela y obtener un desempeño apropiado y eficaz.

Posteriormente la herramienta podría ser el punto de partida en el camino hacia la detección de infractores de tránsito de forma automatizada, lo cual fortalecería la seguridad y disminuiría la reiteración de infracciones de tránsito especificadas anteriormente.

4. Objetivos

4.1. Objetivo general

Desarrollar una herramienta para la identificación de caracteres en la placa de un automóvil a partir de una imagen.

4.2. Objetivos específicos

- Evaluar redes convolucionales en el problema de detección de caracteres de una placa vehicular.
- Seleccionar datos de entrenamiento para nuestra red convolucional.
- Implementar un algoritmo que use redes convolucionales para clasificar placas vehiculares.

5. Marco Referencial

5.1. Marco Teórico

Para la implementación de la herramienta que permita identificar placas vehiculares, es necesario hacer uso de la inteligencia artificial, más específicamente introducirnos en la visión por computador, con ayuda de las redes neuronales convolucionales. Por tal motivo debido a que la aplicación de la herramienta será en un sistema embebido es necesario que el procesamiento sea en tiempo real.

5.2. Antecedentes

Las técnicas de visión por computador han madurado lo suficiente en los últimos veinte años. Los primeros sistemas prácticos de visión por computador se basaron en imágenes binarias (blanco y negro) que se procesaban por bloques, ventanas o píxeles. Rápidamente los algoritmos alcanzaron un nivel de desarrollo en el cual era posible reconocer el contorno del objeto y su posición en la imagen. Sin embargo es obvio que este tipo de tecnología no era lo suficientemente robusta para formar parte de la industria. Los clientes demandaban sistemas que pudieran operar por meses sin necesitar servicio, en condiciones de luz variable, sin importar la textura de los objetos y una velocidad de operación alta.

El siguiente paso en el desarrollo de la visión por computador fue la introducción de los sistemas de intensidad de gris. En ésta tecnología cada elemento de la imagen, o píxel, se representa con un número proporcional a la intensidad de gris del elemento. La principal ventaja de ésta técnica es que puede corregir las variaciones locales de iluminación. Los sistemas de intensidad de gris pueden operar en cualquier tipo de iluminación, ya que pueden encontrar el contorno de los objetos buscando cambios en los valores de intensidad de los píxeles.

Los sistemas de visión por computador de vanguardia operan sobre estructuras en lugar de píxeles. Estos sistemas requieren de procesadores con capacidad de procesar una gran cantidad de datos recibidos en la entrada del sistema y la calidad digital de las imágenes. Un procesador típico utiliza una arquitectura de alta velocidad en paralelo y opera a velocidades de cerca de mil millones de cálculos sobre las imágenes por segundo. Los sistemas de visión por computador utilizan las técnicas de reconocimiento de imágenes para interpretar las propiedades de una imagen, compararla con su base de datos e identificarla como un objeto conocido. Esta tecnología y otros métodos básicos de procesamiento de imágenes se aplican actualmente en tareas como identificación de caracteres, diagnóstico médico, tecnología del espacio, detección de fisuras en la producción de vidrio, tratamiento de residuos sólidos urbanos, reconocimiento de huellas digitales, conversión texto-voz, la clasificación de cuerpos celestes, además de otras relacionadas con la robótica, como la neutralización de explosivos, los análisis sanitarios o el conteo de árboles a partir de fotografías aéreas.

5.3. Bases Teóricas

- *Visión por computador:*

La visión por computador es un campo de la inteligencia artificial que nos permite analizar imágenes obtenidas por medios digitales. Normalmente luego de obtener la imagen se realizan algunas acciones o procesos sobre ella con la finalidad de extraer características de importancia para nosotros según nuestro tema de estudio, todo esto busca poder realizarse de manera autónoma y automatizada, para que pueda ejecutarse en tiempo real o sin la intervención de una persona.

- *Deep Learning:*

El Deep Learning lleva a cabo el proceso de Machine Learning usando una red neuronal artificial que se compone de un número de niveles jerárquicos. En el nivel inicial de la jerarquía la red aprende algo simple y luego envía esta información al siguiente nivel. El siguiente nivel toma esta información sencilla, la combina, compone una información algo un poco más compleja, y se lo pasa al tercer nivel, y así sucesivamente.

Continuando con el ejemplo del gato, el nivel inicial de una red de Deep Learning podría utilizar las diferencias entre las zonas claras y oscuras de una imagen para saber dónde están los bordes de la imagen. El nivel inicial pasa esta información al segundo nivel, que combina los bordes construyendo formas simples, como una línea diagonal o un ángulo recto. El tercer nivel combina las formas simples y obtiene objetos más complejos como óvalos o rectángulos. El siguiente nivel podría combinar los óvalos y rectángulos, formando barbas, patas o colas rudimentarias. El proceso continúa hasta que se alcanza el nivel superior en la jerarquía, en el cual la red aprende a identificar gatos.

- *Redes Neuronales Convolucionales:*

Las redes neuronales convolucionales son muy parecidas a las redes neuronales ordinarias como el perceptrón multicapa, ya que se compone de neuronas con pesos y sesgos que aprenden, usando sus entradas para realizar un producto escalar y luego aplicar una función de activación. Por tanto la diferencia principal de las redes neuronales convolucionales con las ordinarias es que acá suponemos explícitamente que las entradas son imágenes, lo que nos permite codificar ciertas propiedades en la arquitectura; permitiendo ganar en eficiencia y reduciendo la cantidad de parámetros en la red. Principalmente estas redes actúan en problemas que las ordinarias no escalan bien, por ejemplo cuando usamos imágenes de mucha definición.

Estas redes trabajan modelando de forma consecutiva pequeñas piezas de información, y luego combinando esta información en las capas más profundas de la red. Una manera de entenderlas es que la primera capa intentará detectar los bordes y establecer patrones de detección de bordes. Luego, las capas posteriores tratarán de combinarlos en formas más simples y finalmente, en patrones de las diferentes posiciones de los objetos, iluminación,

escalas, etc. Las capas finales intentarán hacer coincidir una imagen de entrada con todos los patrones y arribar a una predicción final como una suma ponderada de todos ellos. De esta forma las redes neuronales convolucionales son capaces de modelar complejas variaciones y comportamientos dando predicciones bastantes precisas.

Estructura:

Básicamente está construida a partir de tres tipos de capas:

- *Capa convolucional:*

En esta capa se aplica algo llamado convolución que básicamente recibe la imagen como entrada y aplica sobre ella un filtro o “kernel” que se usa normalmente para extraer características necesarias o indispensables para nuestro cálculo, ya que normalmente no se necesita todo lo que trae la imagen, entonces con este filtro podemos extraer solo bordes, o eliminar el color y dejar la imagen en escala de grises, entre muchas más cosas, la finalidad de este procedimiento o de esta operación es reducir el tamaño de los parámetros.

- *Capa de reducción o “pooling”:*

Esta capa siempre está ubicada luego de una capa convolucional, con el fin de eliminar elementos innecesarios o simplemente obtiene sólo los valores más representativos y por tal motivo reduce las dimensiones espaciales (ancho por alto) del volumen de entrada para la siguiente capa.

- *Capa clasificadora totalmente conectada:*

Al final de las capas convolucional y de reducción, las redes utilizan generalmente capas completamente conectados en la que cada píxel se considera como una neurona separada al igual que en una red neuronal regular. Esta última capa clasificadora tendrá tantas neuronas como el número de clases que se debe predecir.

5.4. Marco Geográfico

El departamento de Risaralda con una extensión de 365.300 ha, se encuentra en el sector central de la región andina centro occidente del país, en medio de dos grandes polos de desarrollo económico (departamento de Antioquia al norte y el Valle del Cauca al sur, extendiéndose entre la cordillera central y occidental); cuyas laderas descienden hacia el Río Cauca; también limita con los departamentos de Caldas al nororiente, Tolima al oriente, Quindío al sur y Chocó al occidente.

Pereira que es la capital de Risaralda cuenta con varias prestigiosas universidades entre ellas la Universidad Tecnológica de Pereira, donde se alberga un clúster que en una breve definición trata de un conglomerado de computadores que se conectan entre sí por medio de una red, haciendo parecer ante el usuario final un solo equipo de cómputo; habiendo establecido la definición, se utilizará

el clúster para la fase de entrenamiento de la red neuronal convolucional.

En los programas de estudio presentados por la Universidad Tecnológica de Pereira se encuentra de forma mas especifica el programa de Ingeniería de Sistemas y Computación (ISC), donde en dicho programa se encuentra un grupo de investigación llamado Sirius, este a su vez tiene una rama de investigación acerca de Sistemas Inteligentes de Transporte (ITS) y es allí donde se aborda el problema de reconocimiento óptico de caracteres en placas vehiculares.

Espacio Temporal

La delimitación temporal del proyecto estará sujeta al tiempo que se demore desde la fase de entrenamiento hasta que se obtengan resultados concisos.

5.5. Marco Jurídico

A continuación se presentan un conjunto de leyes que regulan las placas vehiculares.

Placas Vehiculares

■ Artículo 43. Diseño y elaboración

Corresponde al Ministerio de Transporte diseñar y establecer las características y ficha técnica de la placa única nacional para los vehículos automotores, asignar sus series, rangos y códigos, y a las autoridades de tránsito competentes o a quien el Ministerio de transporte autorice, su elaboración y entrega. Asimismo, el Ministerio de Transporte reglamentará lo referente a la placa que deberán tener los vehículos que ingresen en el país por programas especiales o por importación temporal [1].

■ Artículo 44. Clasificación

Las placas se clasifican, en razón del servicio del vehículo, así: De servicio oficial, público, particular, diplomático, consular y de misiones especiales [1].

■ Artículo 45. Ubicación

Los vehículos automotores llevarán dos (2) placas iguales: una en el extremo delantero y otra en el extremo trasero.

Ningún vehículo automotor matriculado en Colombia podrá llevar, en el lugar destinado a las placas, distintivos similares a éstas o que las imiten, ni que correspondan a placas de otros países, so pena de incurrir en la sanción prevista en este Código para quien transite sin placas. Estas deben estar libres de obstáculos que dificulten su plena identificación [1].

5.6. Marco Conceptual

Descripción de los conceptos, los cuales son fundamentales para el entendimiento de este proyecto

- **CPU** (*Unidad central de procesamiento*)

Es el hardware dentro de un ordenador u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

- **GPU** (*Unidad de procesamiento gráfico*)

Es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas

- **ALPR** (*Reconocimiento automático de matrículas*)

Es un método de vigilancia en masa que utiliza reconocimiento óptico de caracteres en imágenes para leer las matrículas de los vehículos.

- **CNN** (*Redes neuronales convolucionales*)

Es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico.

6. Diseño Metodológico

La metodología propuesta para el desarrollo de la herramienta se divide en cinco aspectos fundamentales: *Levantamiento de información y Dataset*, *Entrenamiento de la red*, *Implementación del algoritmo*, *Inferencia en la red* y *Análisis de los resultados*.

La etapa de *Levantamiento de información y Dataset* consiste en realizar una consulta lo suficientemente amplia para entender el funcionamiento de una red convolucional, además de la arquitectura que se usó y el conjunto de datos necesario para entrenar la red.

A partir de toda la información adquirida y el conjunto de entrenamiento, se procedió al *Entrenamiento de la red*, que básicamente es enseñarle a la red neuronal convolucional a detectar los caracteres presentes en una placa vehicular. Con la red apta para detectar los caracteres, se realizará la *Implementación* del algoritmo que facilita todo el proceso desde que se obtiene la imagen de la placa vehicular hasta los posibles resultados de la *Inferencia de la red* y de éste modo realizar el *Análisis de los resultados* que permitirán verificar la efectividad de la herramienta.

Para verificar el cumplimiento de los objetivos se usará como métrica la cantidad de placas reconocidas con éxito sin cometer ningún error y de esta forma establecer un porcentaje de eficacia de la red convolucional . Se espera obtener un porcentaje de eficacia superior al 70 %.

Adicionalmente se realizará una comparativa de velocidad en la etapa de *Entrenamiento de la red*, para verificar los tiempos de ejecución en CPU frente a GPU y poder concluir si existe aceleración alguna.

7. Delimitación del alcance

El planteamiento de este proyecto se realizó para generar y desarrollar una herramienta que permite el reconocimiento óptico de caracteres en placas vehiculares privadas de Colombia, a partir de una imagen.

Dicha imagen debe ser una imagen frontal de la placa vehicular sin ninguna inclinación, ésta será la entrada para la herramienta que le proporcionará como resultado en un archivo de texto, los caracteres reconocidos en la placa vehicular.

8. Implementación

8.1. Levantamiento de Información

8.1.1. Materiales y Métodos

Los algoritmos fueron procesados en un computador Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz de arquitectura x86-64, con 8 CPUs, L1d cache y L1i cache de 32K, L2 cache de 256K y L3 cache de 8192K; un Xeon E3-1200 v3/4th Gen Core Processor Integrated Graphics Controller de Intel Corporation, con una tarjeta gráfica NVIDIA Corporation GM204 [GeForce GTX 980] el cual cuenta con 2048 CUDA Cores, 1126 MHz en el reloj base, un ancho de banda de memoria de 224 GB/s y una configuración de memoria estándar de 4 GB [2].

Se utilizó la tecnología CUDA durante la fase de *Entrenamiento de la red convolucional*, la librería de OpenCV para el tratamiento de las imágenes (Segmentación de imágenes) y el lenguaje de programación python a través del framework PyTorch.

■ Python

Python es un lenguaje de programación interpretado de propósito general, multiplataforma y multiparadigma (imperativo, orientado a objetos y funcional). Se trata de un lenguaje fuertemente tipado, es decir, no se puede asignar a una variable un valor de tipo distinto al inicial sin una conversión explícita, siendo a su vez este tipado dinámico: el tipo de las variables se determina en tiempo de ejecución en función del valor asignado.

Se escogió este lenguaje por su legibilidad en el momento de escribir o entender el código utilizado para la realización de los diferentes algoritmos, también por su gran variedad de librerías que son de gran ayuda, como por ejemplo NumPy.

■ OpenCV

OpenCV es una librería de visión artificial desarrollada por Intel y actualmente publicada bajo licencia BSD, que cuenta con más de medio millar de algoritmos optimizados para realizar las principales tareas de visión artificial, como el procesamiento de imágenes, detección de características o reconocimiento de objetos. La librería cuenta además con diferentes algoritmos de aprendizaje de máquina como máquinas de soporte de vectores (SVM), Naïve Bayes o KNN entre otros.

Está escrita en C++, es multiplataforma y cuenta con interfaces para trabajar con lenguajes como Java o Python. La gran cantidad de algoritmos disponibles, su velocidad de ejecución y la extensa comunidad de usuarios de que dispone, hacen de OpenCV una herramienta indispensable para desarrollar sistemas de visión artificial basados en software de código abierto.

- **Pytorch**

Pytorch es un framework de Python que permite el crecimiento rápido del Deep Learning, con una fuerte aceleración de la GPU. La característica principal de Pytorch es que utiliza grafos computacionales dinámicos. Todos los frameworks de Deep Learning utilizan un lenguaje, como Python en este caso, y un grafo computacional. Esto funciona así por cuestiones de eficiencia y optimización, pues un grafo computacional corre en paralelo en la GPU [3].

A pesar de que en la actualidad existen otros framework que también trabaja sobre Python como lo es el caso de Tensorflow y que este tiene una capacidad de que el programador no tenga la necesidad de manejar directamente los tensores que se va a utilizar. Pytorch provee un soporte para la ejecución en tarjetas gráficas (GPU) utilizando CUDA.

- **CUDA**

CUDA es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema [4].

Uno de los aspectos más importantes de esta arquitectura es resolver problemas que conlleva a una gran complejidad y carga computacional en el momento de ejecutarse sobre CPU, con la de las tarjetas gráficas (GPU) se pretende resolver de forma más eficiente dichos problemas.

8.1.2. Arquitectura Red Neuronal Convolucional

La arquitectura de la red convolucional que se usó es denominada LeNet desarrollada por Yann LeCun, dicha red está compuesta por varias capas que implementan la extracción de funciones o características y posteriormente se realiza la clasificación de la imagen [9].

La imagen de entrada que alimenta una primera capa convolucional, ésta extrae las funciones o características relevantes de la imagen de entrada, en el siguiente paso se realiza la agrupación, que básicamente es reducir las dimensiones de las funciones del paso anterior, como por ejemplo una reducción del tamaño, ésta reducción se hace mientras se agrupa información importante que generalmente se ejecuta a través de una técnica denominada agrupación máxima. Estos dos pasos se realizan nuevamente y el resultado es la entrada de un perceptrón multicapa completamente conectado.

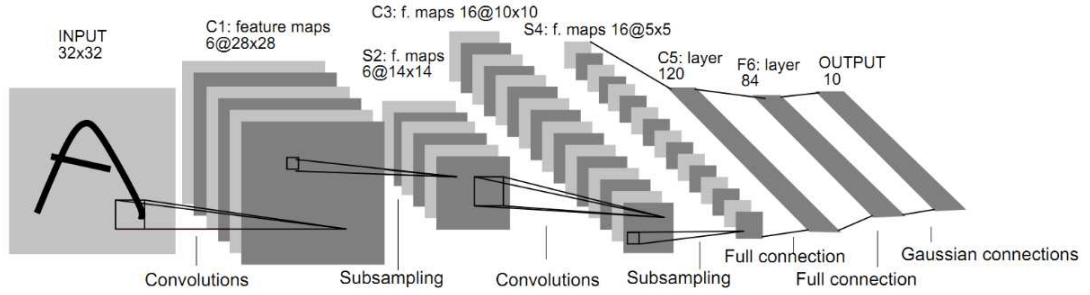


Figura 8.1: Arquitectura LeNet

8.1.3. Dataset

Para el entrenamiento de la red convolucional se hizo uso del conjunto de datos Chars74k creado por T. de Campos y M. Varma de Microsoft Research India y el cual se puede descargar de forma gratuita en la página [surrey](#). Actualmente éste dataset cuenta con una recopilación de alrededor de 64 clases, en las cuales están los números arábigos del 0 al 9 y las letras del abecedario en inglés tanto en minúsculas como en mayúsculas, también se puede encontrar 3 tipos de imágenes, en primer lugar con un total de 7705 imágenes de caracteres en situaciones de la vida real como letreros, el segundo tipo consta de 3410 imágenes de caracteres escritas a mano y por último en el tercer tipo se tiene alrededor de 62292 imágenes de caracteres sintetizados de fuentes de computadora [5].

En este caso y teniendo en cuenta lo antes mencionado, se utilizó el tercer tipo donde no se incluyeron aquellas imágenes que contenían el abecedario inglés en minúsculas, ya que para el problema abordado no se encontrarían caracteres en minúsculas, quedando con un total de 36540 imágenes de entrenamiento y alrededor de 1015 imágenes por clase.

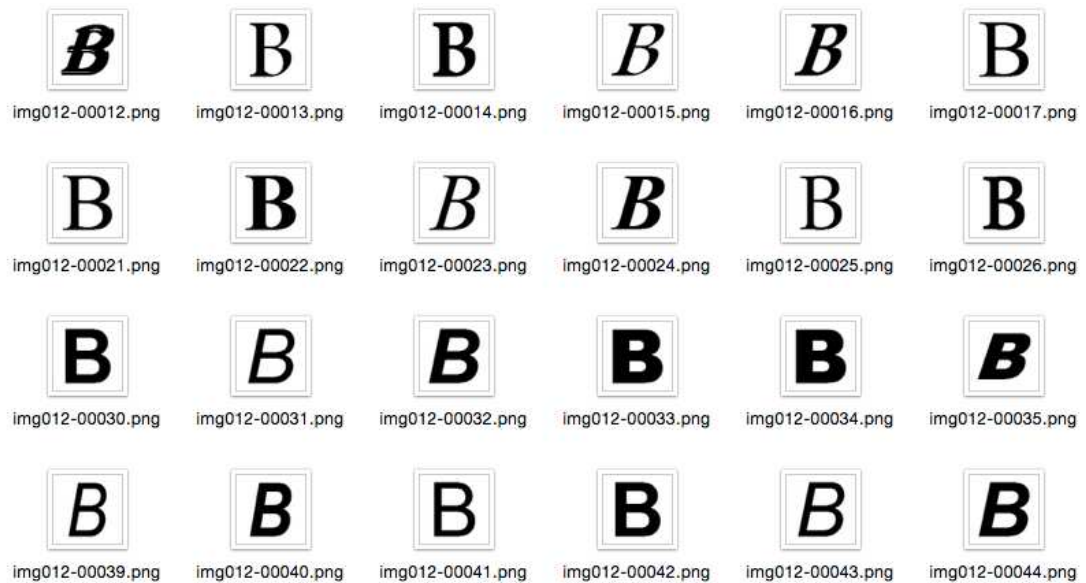


Figura 8.2: Ejemplo dataset Chars74k

8.1.4. Backpropagation

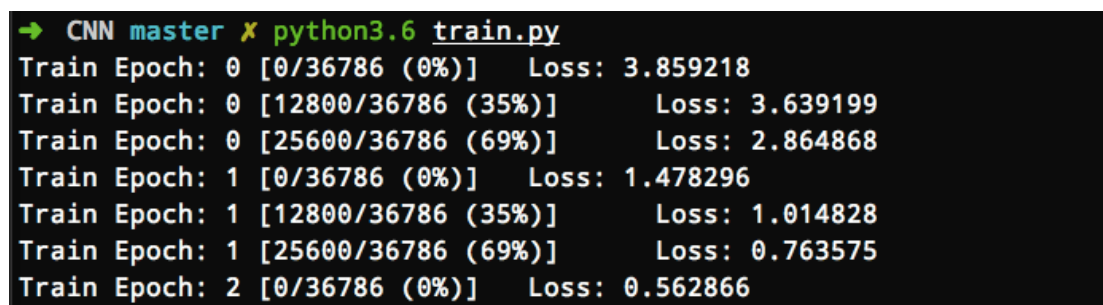
Este método acerca del calculo del gradiente es utilizado en algoritmos de aprendizaje supervisado para entrenar redes neuronales, Utilizando dos fases o también se puede llamar ciclo de propagación, básicamente se aplica un patrón de entrada como estímulo y este se propaga desde la primera capa a través de las siguientes capas de la red hasta generar una salida. Cuando la salida da error, este es propagada hacia atrás desde la capa de salida hasta la neuronas de la capa ocultas involucradas en esa salida; este procedimiento se repite hasta que todas neuronas de la red hayan recibido una señal de error.

8.2. Entrenamiento

En este proceso básicamente se combinan el algoritmo backpropagation con el método del descenso del gradiente, el cual es un problema de optimización en el que iterativamente se buscan los valores de los parámetros (pesos y bias de la red) que hacen que el valor de la función de error sea mínimo, por tanto se decidió utilizar una función muy habitual en el diseño de redes neuronales convolucionales, denominada *entropía cruzada* que nos permitió medir que tan buena será la red convolucional en la clasificación de imágenes.

Por otro lado en el método de descenso del gradiente se usó la versión estocástica o incremental, que consiste en hacer el cálculo iterativo del gradiente para un subconjunto o batch de los datos de entrenamiento e ir actualizando los pesos de la red mediante la propagación de errores hacia atrás, sin esperar a procesar completamente el conjunto de datos de entrenamiento.

En esta etapa el modelo se entrenó durante 10 épocas, imprimiendo el número de la época o etapa en la que se encontraba el entrenamiento y la pérdida que se iba generando, como se observa a continuación:



```
→ CNN master X python3.6 train.py
Train Epoch: 0 [0/36786 (0%)] Loss: 3.859218
Train Epoch: 0 [12800/36786 (35%)] Loss: 3.639199
Train Epoch: 0 [25600/36786 (69%)] Loss: 2.864868
Train Epoch: 1 [0/36786 (0%)] Loss: 1.478296
Train Epoch: 1 [12800/36786 (35%)] Loss: 1.014828
Train Epoch: 1 [25600/36786 (69%)] Loss: 0.763575
Train Epoch: 2 [0/36786 (0%)] Loss: 0.562866
```

Figura 8.3: Entrenamiento

8.3. Algoritmo

El siguiente esquema resume el funcionamiento de la herramienta propuesta para el reconocimiento óptico de caracteres en placas vehiculares:

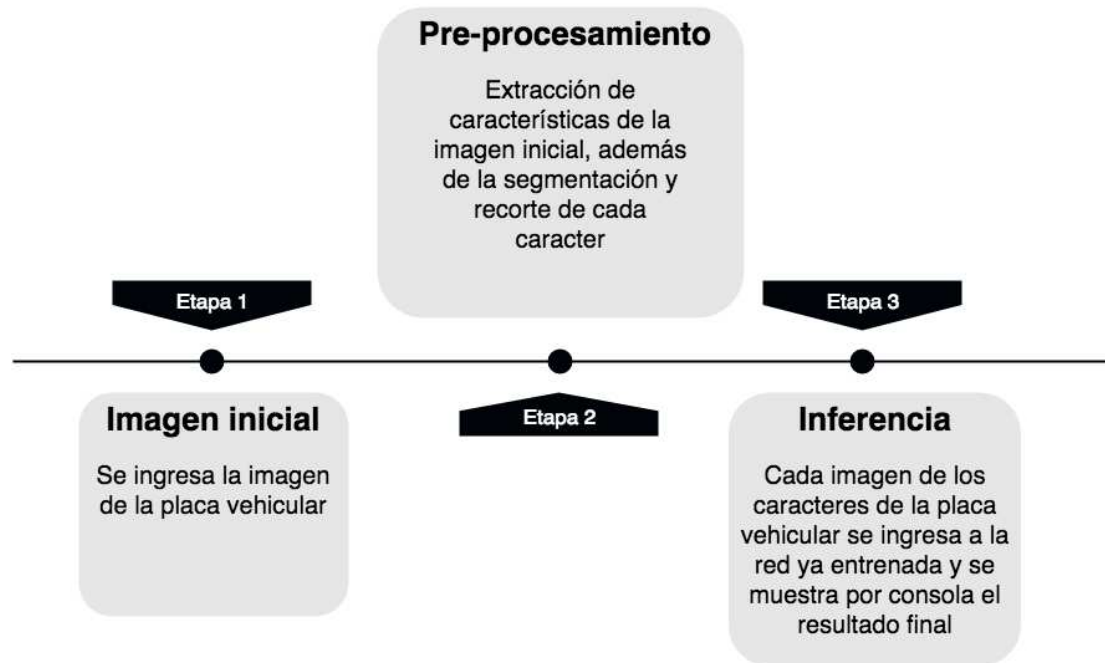


Figura 8.4: Algoritmo

Como se puede observar en el esquema, el algoritmo consta de 3 etapas las cuales serán profundizadas a continuación:

8.3.1. Imagen inicial

En esta etapa se realiza el ingreso de la imagen al sistema, las cuales serán imágenes frontales de placas vehiculares colombianas.



Figura 8.5: Imagen inicial

Una vez ingresada, se procede a realizar el procesamiento de la misma y posteriormente determinar la placa vehicular a la que hace referencia.

8.3.2. Pre-procesamiento

Para el correcto funcionamiento de la herramienta y teniendo en cuenta diferentes aspectos para el procesamiento de imágenes, es necesario realizar un conjunto de pasos o procesos a la imagen inicial, ya que se debe asegurar que su estructura y canales sean indicados, además de eliminar el ruido presente en ella y extraer solo las regiones necesarias.

- **Conversión a escala de grises**

Las imágenes iniciales son imágenes de placas vehiculares colombianas, éstas imágenes son a color y de tamaño variable, por tal motivo se deben adaptar o pre-procesar para que sean aptas en el sistema. Inicialmente basados en un caso base de la visión por computador, en el cual recomiendan convertir las imágenes a escala de grises, ya que este proceso permite facilitar el cálculo u operaciones que vayan a realizar con dichas imágenes porque la cantidad de bits para representarla disminuye, esto ocurre porque las imágenes a color tienen tres canales de tonalidad, en cambio la de escala de grises solo uno.



Figura 8.6: Imagen convertida a escala de grises

- **Segmentación de la imagen**

Con la finalidad de extraer las características necesarias para continuar con el proceso, se usó el thresholding para segmentar la imagen, esto lo que hace es reemplazar cada pixel de la imagen por uno negro o blanco, dependiendo la intensidad que tenga el pixel en la imagen, ésta intensidad es comparada con una constante de threshold que en nuestro caso es 85. Éste proceso se implementó haciendo uso de opencv y su método de segmentación THRESHOLD_BINARY [7], el cual está representado con la siguiente fórmula:

$$dst(x, y) = \begin{cases} 255 & \text{si } src(x, y) > 85 \\ 0 & \text{de otro modo} \end{cases}$$

Con lo cual se obtuvo el siguiente resultado:



Figura 8.7: Thresholding

■ Determinando regiones de interés

Luego de tener los caracteres de la placa resaltados claramente, se debe determinar que región representan cada uno de ellos dentro de la imagen, para esto se buscó los contornos que podrían representar un carácter, el problema es que los valores de los contornos que podrían representar un carácter deben ser obtenidos a prueba y error, ya que las placas son de tamaño variable, luego de muchos intentos se logró determinar que solo los contornos que cumplan con la siguiente condición podrían representar un carácter, cabe resaltar que antes hay que encontrar el bounding box [6] o cuadro delimitador de cada contorno para obtener la altura y anchura de cada uno de ellos

Por tanto un contorno representa un carácter si:

$$85 < altura, anchura < 300$$



Figura 8.8: Área de interés

Luego de obtenidas las seis regiones de interés de nuestra imagen, pasamos a recortar cada una de ellas y redimensionarlas a 32x32 píxeles, ya que es el tamaño establecido en la red convolucional para su entrada.

- **Centrando el carácter**

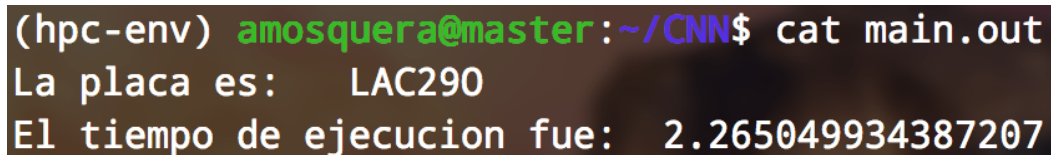
Al obtener las regiones de interés, se logró observar que cada una representaba un dígito, pero no contaban con suficiente borde, para que pudieran ser analizados por la red convolucional, ya que debían estar centrados para obtener un resultado satisfactorio, por tanto se tuvo que centrar el carácter. Esto se realizó por medio de OpenCV, luego de obtener la relación de aspecto de la imagen, se calculó en base a eso cuántos píxeles se podría añadir a los bordes sin deformar la imagen y luego, haciendo uso de `copyMakeBorder` [8] que es una función de OpenCV se logró añadir el borde a la imagen, dándole el color y el tamaño en píxeles.

- **Enviando a Inferencia**

En esta etapa básicamente se guardaron las imágenes para que en la fase de Inferencia fuera posible acceder a ellas y realizar el proceso de reconocimiento de cada carácter en la red convolucional ya entrenada.

8.3.3. Inferencia

Luego de entrenada la red, se pasó a unir los procesos para realizar el reconocimiento óptico de los caracteres. Las imágenes que representan cada uno de los caracteres extraídos de la imagen, se almacenaron en una carpeta, por lo tanto era necesario traer a la red uno por uno, para que ella pudiera determinar que carácter representaba cada una de las imágenes, luego se concatenaron los caracteres y así se pudo determinar totalmente la placa vehicular que estaba representada en la imagen inicial.



```
(hpc-env) amosquera@master:~/CNN$ cat main.out
La placa es: LAC290
El tiempo de ejecucion fue: 2.265049934387207
```

Figura 8.9: Resultado

9. Pruebas

Con el fin de evaluar la efectividad de la red convolucional ya entrenada y haciendo uso del algoritmo mencionado anteriormente, se realizó un conjunto de pruebas que abarcan alrededor de 120 imágenes de placas vehiculares, 20 de ellas fueron de placas reales y las 100 restantes fueron generadas automáticamente con la ayuda de una herramienta web [13], a continuación se podrá observar la herramienta y su utilización:

CUSTOMIZE YOUR PLATE!

PLEASE ENTER YOUR CUSTOM LETTERS/NUMBERS INTO THE BOX BELOW

XRT859

SELECT QUANTITY

1

ADD TO CART

Free ULTRA Shipping
Included in the USA

AVAILABLE CHARACTERS LEGEND

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
V	W	X	Y	Z	1	2	3	4	5	6	7	8	9	0	}	\	/	-	+	.	[
V	W	X	Y	Z	1	2	3	4	5	6	7	8	9	0							

Figura 9.1: Herramienta generadora de placas vehiculares

Para generar la placa vehicular, basta con ingresar los dígitos alfa-numéricos que se necesiten plasmar en ella y la herramienta automáticamente la generará, permitiendo fácilmente su descarga.

Posteriormente la imagen obtenida se ingresó al sistema, el cual nos arrojó un resultado que fue el insumo para el cálculo de la eficacia de la herramienta, como se mencionó anteriormente.

Adicionalmente se midió el tiempo de ejecución de la etapa de Entrenamiento en CPU y GPU, con el fin de verificar si la GPU es más rápida que la CPU en procesos costosos (computacionalmente hablando).

10. Resultados

Luego de realizadas las pruebas correspondientes, teniendo en cuenta que se realizaron en 120 imágenes, con 100 imágenes generadas y 20 de placas vehiculares reales, se obtuvieron los siguientes resultados:

Tipo de placa	Total imágenes	Aciertos	Fallos
Generadas	100	70	30
Reales	20	20	0

Cuadro 10.1: Resultados

Al analizar la evaluación del sistema con las imágenes reales, podemos observar que la herramienta fue realmente eficaz, ya que se obtuvo 100 % de aciertos en el reconocimiento del contenido de las placas vehiculares.

Por otro lado, en el caso de las imágenes generadas se obtuvo una eficacia del 70 %, pero con fallos por uno, dos y en ocasiones 3 caracteres que dentro de lo esperado podría ser razonable, ya que al realizar el reconocimiento hay una confusión entre caracteres muy similares, como lo son la letra S y el número 5, además de la letra O y el número 0; otro caso en específico ocurrió con las letras Q y O, ya que la diferencia entre ellas en la tipografía usada en las placas vehiculares colombianas es poco perceptible.

Se pudo definir que el porcentaje de eficacia de la herramienta en general, fue del 75 %.

En la tabla Tabla 10.3 se puede observar una lista completa de los errores cometidos por la herramienta en las matrículas generadas.

Matricula	OCR	Errores
CWE093	CWEO93	1
UET881	UETBB1	2
OBH409	OBH4O9	1
CVY000	CVYOOO	3
WHI000	WHIOOO	3
FAK579	F4KS79	1
NSP000	NSPOOO	3
BSI859	BSIB59	1
QGU210	OGU21O	2
LYG335	LYG33S	1
HBR687	HBR6B7	1
BUR685	8UR685	1
POQ342	POO342	1
NNS823	NN5B23	2
QMX593	OMXS93	2
TTY582	TTYS82	1
PLO074	PLOO74	1
IOO001	IOOOO1	2
PQS324	POS324	1
SSW552	SSWSS2	2
QRO559	OROSS9	3
UHB763	UH8763	1
KUU990	KUU99O	1
JHG651	JHG6S1	1
VDI590	VDIS90	1
SSU980	SSU98O	1
XLM012	XLMO12	1
UTO550	UTOSSO	3
PKI011	PKIO11	1
RTQ042	RTOO42	2

Cuadro 10.2: Errores en placas generadas

Por otro lado tras la ejecución de la fase de entrenamiento de la red neuronal convolucional sobre GPU y CPU, se pudo observar que la GPU es más óptima en este proceso, gracias a su capacidad de procesamiento en paralelo. A continuación se puede verificar el resultado:

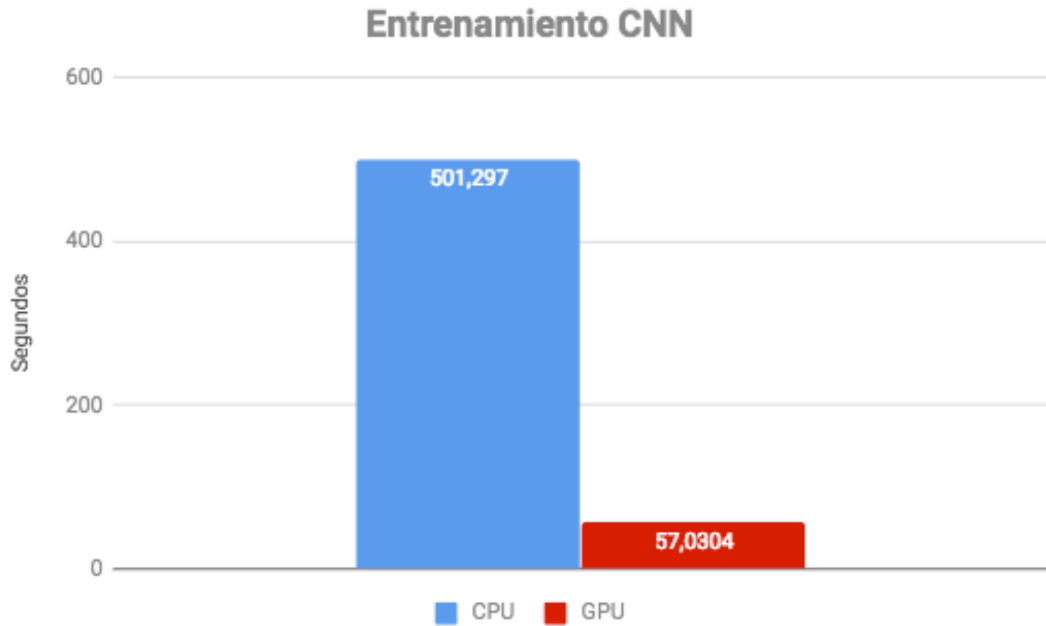


Figura 10.1: Tiempo de entrenamiento

La mejora del tiempo de ejecución del entrenamiento de la red convolucional se da debido a que se cuenta con un dataset de entrenamiento de alrededor de 36.000 imágenes las cuales implican tener muchos recursos para su procesamiento y posteriormente su entrenamiento en la red, además que las redes convolucionales están diseñadas específicamente para el procesamiento de imágenes y sus propiedades, haciendo más rápido el uso de la GPU por encima de la CPU, adicionalmente el uso del paralelismo con CUDA para la GPU la hace especial para este tipo de procesamiento y mucho mas óptima en tiempo de ejecución, siendo 8 veces mas rápida la GPU que la CPU como se observa a continuación:

Aceleración Entrenamiento
8,7899

Cuadro 10.3: Tabla Aceleración Entrenamiento CNN

11. Problemas Encontrados

11.0.1. Conjunto de datos en Pytorch

Pytorch tiene un módulo dedicado a la carga de conjunto de datos, pero para el problema que nosotros intentábamos resolver, existía un conjunto de datos incluido en el Framework, el cual representa una extensión o variación de MNIST que es un conjunto de imágenes de dígitos numéricos y es normalmente el primer acercamiento que se tiene cuando se empieza a trabajar con redes neuronales convolucionales, esta variación de dicho conjunto de datos es llamado Extended-MNIST [11], ya que adicional a los dígitos numéricos también añaden letras mayúsculas y minúsculas. Al usar Pytorch la mayoría de los conjuntos de datos son transformados dependiendo la estructura que tenga la red convolucional, por lo tanto frecuentemente las transformadas más usadas son las conversiones de las imágenes a tensores y la normalización de sus datos, ya que esto permite un mejor cálculo a la hora de entrenar la red, esto nos dificultó mucho el proceso, debido a la poca información que encontrábamos sobre la desviación estándar necesaria para la normalización de los valores y por consiguiente las adaptaciones necesarias a la arquitectura de la red que usamos, en base a esta problemática y tras muchos intentos fallidos de poder utilizar este conjunto de datos, optamos por construir nuestro propio loader o cargador de archivos, para tener un mayor control de nuestro sistema, lo cual se realizó en base a una red construida anteriormente [10] y haciendo uso del conjunto de datos chars74k, que no es tan extenso como lo es E-MNIST pero para la práctica fue mucho más funcional.

11.0.2. Orden de las regiones de interés

Cuando determinamos las regiones de interés o básicamente las regiones que podrían representar un carácter en nuestra imagen, surgió una dificultad con el orden en que se entregaban estas regiones, ya que se entregaban de manera aleatoria, dependiendo de factores de ubicación y tamaño, por lo tanto tuvimos que realizar un ordenamiento en base a la posición en el eje x del bounding box o rectángulo delimitante de nuestra región de interés, esto se realizó en base a [6].

12. Conclusiones

- Hacer uso del paralelismo y las unidades de procesamiento gráfico (GPUs), reduce considerablemente el tiempo de entrenamiento de una red neuronal convolucional, ya que gracias a su gran cantidad de cores o núcleos permiten procesar imágenes mucho más rápido.
- Para realizar un ALPR [12] de placas colombianas con un grado de acierto muy alto, es necesario construir un conjunto de datos bastante extenso y específico, ya que algunos caracteres pueden ser ambiguos para un computador, entre ellos los más propensos a errores son la letra Q, ya que tiende a parecerse mucho a la O, debido a lo poco pronunciada que es su raya inferior y por otro lado está la confusión entre la letra O y el número 0, ya que el tipo de letra que se usa en las placas colombianas no determina una diferencia considerable.
- Este tipo de herramientas pueden ser de gran utilidad en lugares que necesitan de vigilancia constante o simplemente en control de tráfico, pudiendo convertirse en un aporte a la Big Data y el análisis de datos vehiculares.
- El futuro de las redes neuronales convolucionales en el mundo de la visión por computador es bastante prometedor, ya que tienen un desempeño muy alto en la clasificación de objetos, esto gracias a su arquitectura que está totalmente orientada a las imágenes.

Futuros trabajos

Como continuación de este trabajo de grado, existen diversas líneas de investigación que quedan abiertas y en las que es posible continuar trabajando. Durante el desarrollo de este trabajo han surgido problemáticas que se esperan atacar en un futuro.

A continuación se presentan algunos trabajos futuros que pueden desarrollarse como resultado de éste trabajo o que, por exceder el alcance del trabajo, no han podido ser tratados con la siguiente profundidad. Además se sugieren algunos desarrollos específicos para apoyar, mejorar u optimizar el modelo y metodología propuestos, entre ellos se destacan:

- Se podría extender el alcance del trabajo, hacia reconocimiento óptico de caracteres en placas vehiculares de diferente tipo, procedencia, colores o también se podría cambiar los ángulos de la imagen inicial de la herramienta.
- La herramienta podría hacer parte de la construcción de un sistema de detección de infractores de tránsito o para el control y seguimiento de vehículos.
- Para aumentar la eficacia de la herramienta se podría intentar modificar la arquitectura de la red neuronal convolucional o hacer uso de algunas arquitecturas más robustas y potentes como puede ser AlexNET. También se podría usar un conjunto de datos más grande y específico.
- Se puede incluir la herramienta en un software mas robusto, donde no solamente sirva para detectar caracteres de placas vehiculares, sino que también permita localizar la placa vehicular en una imagen y posteriormente realizar el reconocimiento óptico de los caracteres pertenecientes a ella.

Bibliografía

- [1] Código Nacional de Tránsito Terrestre <https://www.colombia.com/actualidad/images/2008/leyes/transito.pdf>
- [2] Referencia del Cluster, GTX 980 Specifications. NVIDIA. Obtenido de <https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-980/specifications/>
- [3] Referencia de Pytorch, obtenida de <https://cleverpy.com/que-es-pytorch-y-como-se-instala/>
- [4] Pagina oficial de CUDA <http://www.nvidia.es/object/cuda-parallel-computing-es.html>
- [5] Datos obtenidos desde la pagina oficial donde podemos descargar gratuitamente el dataset <http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/>
- [6] Recognizing digits with OpenCV and Python <https://www.pyimagesearch.com/2017/02/13/recognizing-digits-with-opencv-and-python/>
- [7] Pagina oficial de OpenCV, <https://opencv.org/>
- [8] Adding borders to your images, <https://docs.opencv.org/2.4.13.4/doc/tutorials/imgproc/imgtrans/copyMakeBorder/copyMakeBorder.html>
- [9] Pagina oficial de LeNet, <http://yann.lecun.com/exdb/lenet/>
- [10] Repositorio de gaurav0651, https://github.com/gaurav0651/emnist/blob/master/train_emnist.ipynb
- [11] Pagina oficial de EMNIST, <https://www.nist.gov/itl/iad/image-group/emnist-dataset>
- [12] Documentación acerca de ALPR, <https://matthewearl.github.io/2016/05/06/cnn-anpr/>
- [13] Herramienta web generadora de placas vehiculares <https://www.licenseplates.tv/colombia-armenia-license-plate-embossed-with-your-custom-number-2.html>

Índice de cuadros

10.1. Resultados	32
10.2. Errores en placas generadas	33
10.3. Tabla Aceleración Entrenamiento CNN	34

Índice de figuras

8.1. Arquitectura LeNet	23
8.2. Ejemplo dataset Chars74k	23
8.3. Entrenamiento	24
8.4. Algoritmo	25
8.5. Imagen inicial	25
8.6. Imagen convertida a escala de grises	26
8.7. Thresholding	27
8.8. Área de interés	27
8.9. Resultado	28
9.1. Herramienta generadora de placas vehiculares	30
10.1. Tiempo de entrenamiento	34

Anexos

Módulo de la Red Neuronal Convolutacional

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import torchvision.datasets as dset
import torchvision.transforms as transforms
import numpy as np
from PIL import ImageOps
import torch.optim as optim
from PIL import Image
import time

# caracteres que reconoceremos en la placa
idx = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
       'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
       'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
       'U', 'V', 'W', 'X', 'Y', 'Z']

# carga de las imagenes en entrenamiento
def pil_loader(path):
    with open(path, 'rb') as f:
        img = Image.open(f)
        return img.convert('L')

# clase que representa nuestra red convolutacional
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 128, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(3200, 2048)
        self.fc3 = nn.Linear(2048, 512)
        self.fc5 = nn.Linear(512, 47)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 3200)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = F.relu(self.fc3(x))
        x = F.dropout(x, training=self.training)
        x = self.fc5(x)
        return x
```



```

# funcion que se encarga del entrenamiento del modelo
def train(epoch, model, optimizer, loader):
    model.train()
    for batch_idx, (data, target) in enumerate(loader):
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = nn.CrossEntropyLoss()
        output = loss(output, target)
        output.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} \t Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(loader.dataset),
                100. * batch_idx / len(loader), output.data.item()))

# funcion que se encargan de reconocer cada caracter
def predict_char(gray, model):
    w = gray.size[0]
    h = gray.size[1]
    gray = gray.convert('L')
    gray = gray.point(lambda x: 0 if x < 180 else 255, '1')
    x = int((w - 1) / 2)
    y = int((h - 1) / 2)
    canvas = Image.new('L', (32, 32), (255))
    canvas.paste(gray, box=(x, y))

    canvas = np.array(canvas)

    test_output = model(Variable(torch.FloatTensor(canvas)
        .unsqueeze(0).unsqueeze(0).data))
    pred = test_output.data.max(1, keepdim=True)[1]
    pred = np.array(pred).squeeze(0).squeeze(0)
    return idx[pred]

```

Módulo de Segmentación y Pre-procesamiento

```
import cv2
import imutils
import operator

# funcion encargada de retornar los bordes que representan las roi
def digit_segmentation(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 85, 240, cv2.THRESH_BINARY)[1]
    cnts = cv2.findContours(thresh.copy(),
        cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)[0]
    digits = {}
    for c in cnts:
        (x, y, w, h) = cv2.boundingRect(c)
        if (w > 90 and w < 300) and (h > 90 and h < 300):
            cv2.rectangle(image, (x - 10, y - 10),
                (x + w + 10, y + h + 10), (255, 0, 0), 5)
            roi = thresh[y : y + h, x : x + w]
            digits[x] = roi

    return digits

# funcion que almacena cada roi recortado
def save_digits(images):
    for idx, image in enumerate(images):
        image = cv2.resize(image, (28, 32),
            interpolation = cv2.INTER_CUBIC)
        image = roi_center(image)
        cv2.imwrite('./Test/test{}.jpg'.format(idx), image)

# funcion para centrar el roi
def roi_center(roi):
    ALTURA_ROI = 28
    ANCHURA_ROI = 32
    ANCHO_BORDE = 4

    relacion_aspecto_roi = float(roi.shape[1]) / float(roi.shape[0])
    nueva_anchura = int((ALTURA_ROI * relacion_aspecto_roi) + 0.5)
    b_top = ANCHO_BORDE
    b_bottom = ANCHO_BORDE
    b_left = int((ANCHURA_ROI - nueva_anchura) / 2)
    b_right = int((ANCHURA_ROI - nueva_anchura) / 2)

    roi_borde = cv2.copyMakeBorder(roi, b_top, b_bottom, b_left, b_right, cv2.
        BORDER_CONSTANT, value=[255, 255, 255])
    roi_trans = cv2.resize(roi_borde, (32, 32))
    return roi_trans

if __name__ == "__main__":
    image_src = cv2.imread("./Placas/placa2.jpg")
    image_src = imutils.resize(image_src, height=500)
    result = digit_segmentation(image_src)
    result_sort = sorted(result.items(), key=operator.itemgetter(0))
    final_result = []
```

```
for r in result_sort:
    final_result.append(r[1])

save_digits(final_result)
```

Módulo de Entrenamiento

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import torchvision.datasets as dset
import torchvision.transforms as transforms
import numpy as np
from PIL import ImageOps
import torch.optim as optim
from PIL import Image
import time
import cnn

if __name__ == "__main__":
    trans = transforms.Compose([transforms.Resize((32,32)), transforms.ToTensor()])
    mydata = dset.ImageFolder('./char74k', transform=trans, loader=cnn.pil_loader)
    loader = torch.utils.data.DataLoader(mydata, batch_size=128, shuffle=True, num_workers=2)

    model = cnn.Net()
    model.cuda()
    optimizer = optim.Adam(model.parameters(), lr=1e-4, eps=1e-4)
    for epoch in range(10):
        cnn.train(epoch, model, optimizer, loader)
    torch.save(model.state_dict(), 'char_recognizer3.pt')
```

Módulo Principal

```
import cv2
import imutils
import operator
import segmentation
import cnn
import torch
from PIL import Image

if __name__ == "__main__":

    model = cnn.Net()
    model.load_state_dict(torch.load("char_recognizer.pt"))

    # descomentar la siguiente linea para usar CUDA
    #model.cuda()

    res = ""
    for i in range(6):
        pil_im = Image.open("./Test/test{}.jpg".format(i))
        res += cnn.predict_char(pil_im, model)
    print('La_placa_es:__', res)
```

Participantes

Ramiro Andrés Barrios Valencia	MSc. Ingeniería de Sistemas y Computación
Anderson Steven Mosquera Mosquera	Candidato: Ing. Sistemas y Computación
Juan Pablo Martínez Rendón	Candidato: Ing. Sistemas y Computación